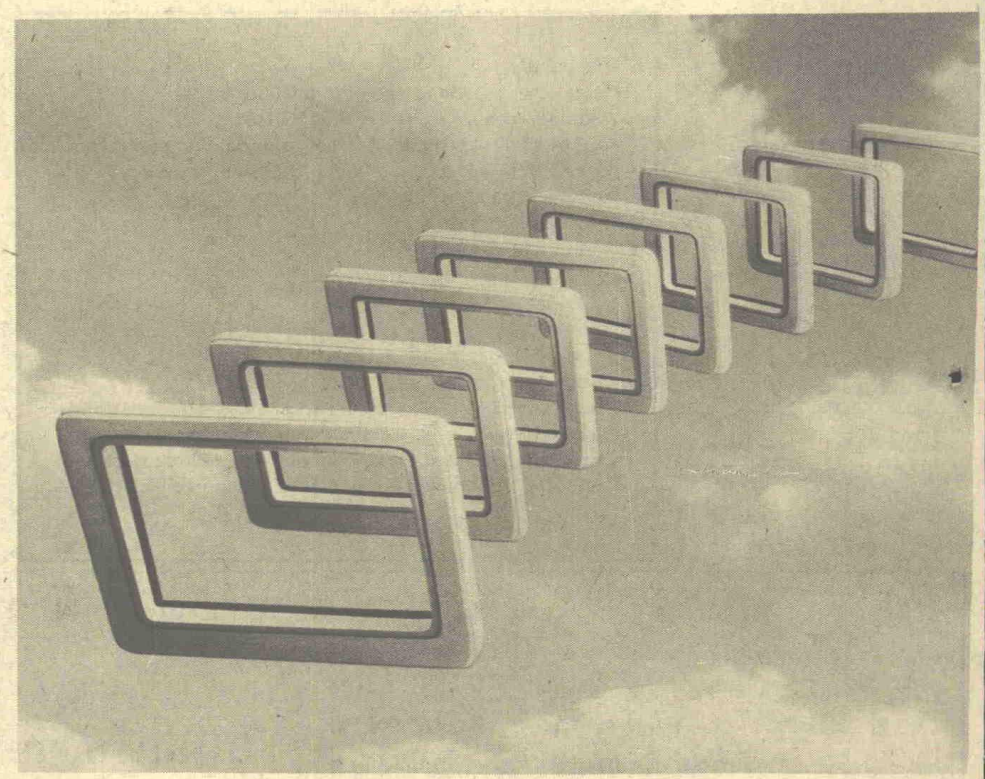
ประสิทธิ์ ออมจินดาทรัพย์

เทคนิคการสร้าง Collection ในเทอร์โบปาสคาล

เป็นการแนะนำ การใช้งานคอลเลคชัน ที่เป็นประโยชน์อย่างมาก ในการทำงาน



บ่อยครั้งที่คุณใช้อะเรย์หรือลิสต์ในการเก็บข้อมูล ถ้าหากข้อมูลของคุณมีจำนวนคงที่และต้องการความเร็วในการเรียกใช้ข้อมูล คุณจะต้องเลือกใช้โครงสร้างข้อมูลแบบอะเรย์ แต่ถ้าหากงานของคุณเป็นงานที่ไม่ทราบถึงจำนวนของข้อมูลที่จะเก็บในอะเรย์ คุณคงต้องการไดนามิกอะเรย์ที่มีความสามารถขยายหรือลดขนาดของตัวมันเองได้ หรือคุณอาจจะใช้ลิงค์ลิสต์มาใช้แก้ปัญหา แต่ว่าทั้งไดนามิกอะเรย์และลิงค์ลิสต์นั้นมีความยุ่งยากและซับซ้อนในการออกแบบโครงสร้างและการออกแบบโพรซีเดอร์ เช่น การ insert, delete, find, update หรือจัดการดาต้าต่าง ๆ

ปัญหาข้างต้นนั้นจะเป็นเพียงปัญหาเล็กน้อย ถ้าหากคุณกำลังใช้เทอร์โบปาสคาล เวอร์ชัน 6.0 ขึ้นไป ในการโปรแกรมผมขอแนะนำให้ท่านผู้อ่านรู้จัก Turbo Vision's low-level objects ที่มีชื่อว่า TCollection ซึ่งตัว TCollection นี้จะมีความสามารถในการสร้างและใช้งานได้ดี กับ low-level structures ในโปรแกรมเทอร์โบปาสคาล

ภายใน Collections

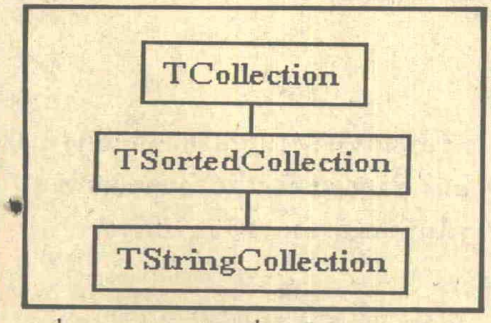
TCollection เป็นออปเจ็กต์ที่บรรจุวิธีการที่จะกระทำการต่าง ๆ ในส่วนของอะเรย์ในพอยน์เตอร์ ซึ่ง TCollection นี้จะมีความสามารถในการจัดการข้อมูลต่าง ๆ ในลิสต์ถึง 23 วิธี โดยในรูปที่ 1 จะแสดง Predefind Pointers และ Method ต่าง ๆ ของ TCollection ที่เทอร์โบวิชันเตรียมให้คุณใช้งาน รายละเอียดต่าง ๆ นั้นสามารถดูได้ใน OBJECTS.INT ที่อยู่ในไดเรคทอรีที่ชื่อ \tp\doc โดยก่อนที่จะใช้งาน TCollection นั้น คุณจะต้องทำการ uses objects ก่อนทุกครั้ง (objects เป็น unit ที่เพิ่มเข้ามาในเทอร์โบปาสคาล 6.0 ขึ้นไป)

TCollection Descendants

ก่อนที่จะทำการสร้างคอลเลคชันกัน ผมขอแนะนำให้รู้จักออปเจ็กต์ใหม่อีก 2 ตัวคือ TSortedCollection และ

```
{ TCollection types }

PItemList = ^TItemList;
TItemList = array[0..MaxCollectionSize - 1] of Pointer;

{ TCollection object }

PCollection = ^TCollection;
TCollection = object(TObject)
  Items: PItemList;
  Count: Integer;
  Limit: Integer;
  Delta: Integer;
  constructor Init(ALimit, ADelta: Integer);
  constructor Load(var S: TStream);
  destructor Done; virtual;
  function At(Index: Integer): Pointer;
  procedure AtDelete(Index: Integer);
  procedure AtFree(Index: Integer);
  procedure AtInsert(Index: Integer; Item: Pointer);
  procedure AtPut(Index: Integer; Item: Pointer);
  procedure Delete(Item: Pointer);
  procedure DeleteAll;
  procedure Error(Code, Info: Integer); virtual;
  function FirstThat(Test: Pointer): Pointer;
  procedure ForEach(Action: Pointer);
  procedure Free(Item: Pointer);
  procedure FreeAll;
  procedure FreeItem(Item: Pointer); virtual;
  function GetItem(var S: TStream): Pointer; virtual;
  function IndexOf(Item: Pointer): Integer; virtual;
  procedure Insert(Item: Pointer); virtual;
  function LastThat(Test: Pointer): Pointer;
  procedure Pack;
  procedure PutItem(var S: TStream; Item: Pointer); virtual;
  procedure SetLimit(ALimit: Integer); virtual;
  procedure Store(var S: TStream);
end;

{ TSortedCollection object }

PSortedCollection = ^TSortedCollection;
TSortedCollection = object(TCollection)
  Duplicates: Boolean;
  constructor Load(var S: TStream);
  function Compare(Key1, Key2: Pointer): Integer; virtual;
  function IndexOf(Item: Pointer): Integer; virtual;
  procedure Insert(Item: Pointer); virtual;
  function KeyOf(Item: Pointer): Pointer; virtual;
  function Search(Key: Pointer; var Index: Integer): Boolean; virtual;
  procedure Store(var S: TStream);
end;

{ TStringCollection object }

PStringCollection = ^TStringCollection;
TStringCollection = object(TSortedCollection)
  function Compare(Key1, Key2: Pointer): Integer; virtual;
  procedure FreeItem(Item: Pointer); virtual;
  function GetItem(var S: TStream): Pointer; virtual;
  procedure PutItem(var S: TStream; Item: Pointer); virtual;
end;
{ TResourceCollection object }
PResourceCollection = ^TResourceCollection;
TResourceCollection = object(TStringCollection)
  procedure FreeItem(Item: Pointer); virtual;
  function GetItem(var S: TStream): Pointer; virtual;
  function KeyOf(Item: Pointer): Pointer; virtual;
  procedure PutItem(var S: TStream; Item: Pointer); virtual;
end;
```

รูปที่ 1 แสดง Predefind Pointer และ Method ของ TCollection



รูปที่ 2 ออปเจ็กต์ใหม่ที่สืบสกุลมาจาก TCollection

TStringCollection ซึ่งทั้ง 2 ตัวนั้นสืบสกุลมาจาก TCollection ดังแสดงในรูปที่ 2

จากแนวความคิดแบบ OOP นั้นข้อดีอย่างหนึ่งคือ TStringCollection จะมีคุณสมบัติเหมือนบรรพบุรุษมันคือ TSortedCollection และในทำนองเดียวกัน TSortedCollection ก็จะมีคุณสมบัติทุกอย่างเหมือนกับ TCollection เช่นกัน

TSortedCollection

จะมีวิธีการค้นหาและจัดเรียงสำหรับรายการ Collection

Compare : ใช้ในการค้นหาและเปรียบเทียบรายการ Collection

Index of : ใช้ค้นหาดัชนีของรายการที่ได้รับ

Insert : ใช้แทรกรายการและจัดการจัดลำดับเรียงความ

Keyof : จะส่งกลับค่าคีย์ของรายการ

Search : จะทำการค้นหาแบบไบนารี (binary search) ในการจัดเรียงของ TCollection และส่งกลับค่าตำแหน่งของรายการ

TStringCollection

จะเก็บแอสกีสตริงที่จัดเรียงใน Collection และง่ายในการใช้งาน

Compare : ใช้เปรียบเทียบสตริง

FreeItem : จัดเก็บและให้หน่วยความจำโดย Collection item string

GetItem : อ่านสตริงจาก Stream ที่กำหนด

PutItem : เขียนสตริงลงบน Stream ที่กำหนด

การจะเลือกใช้ TSortedCollection หรือ TStringCollection ก็ควรจะพิจารณาถึงความเหมาะสมของแต่ละงานด้วย

การสร้าง Collection

ในการสร้าง TCollection, TSortedCollection และ TStringCollection นั้น อันดับแรกคุณจำเป็นจะต้องกำหนดพอยน์เตอร์ที่มีไทพ์เป็น PCollection, PSortedCollection หรือ PStringCollection ซึ่งพอยน์เตอร์นี้จะเป็นตัวแปรโกลบอลหรือโลคอลก็ได้ ต่อไปก็ทำการจองเนื้อที่ว่างสำหรับ Collection object โดยการเรียกใช้คอนสตรัคเจอร์ที่ชื่อ Init ซึ่งมี Syntax คือ

```
Constructor Init(ALimit, ADelta : Integer);
```

โดยที่วิธีการนี้จะมีพารามิเตอร์ 2 ตัวคือ ALimit และ ADelta ซึ่ง ALimit จะเป็นการกำหนดจำนวนของไอเท็มที่คุณต้องการในคอลเลคชัน และถ้าหากคอลเลคชันเกิดขยายเกินจำนวน ALimit แล้ว Collection จะขยายขึ้นอีกเป็นจำนวน ADelta ตัวอย่างการใช้งานดังนี้

```
Var
      ColP : PCollection;
begin
      ColP := Net(TCollection,Init(20,5));
end;
```

ตัวแปร ColP จะถูกกำหนดให้เป็น PCollection และในบรรทัดต่อไปจะเป็นการจองเนื้อที่ 80 ไบต์ (20× sizeof(pointer) ซึ่งขนาดของพอยน์เตอร์ = 4 ไบต์) ให้กับ ColP ซึ่งสามารถบรรจุข้อมูลได้ 30 รายการ และสามารถขยายได้ทีละ 5 รายการถ้าหากว่าคอลเลคชันเกิดเต็ม ซึ่งในการทำงานจริง ๆ คุณควรจะจองเนื้อที่ให้มีขนาดพอดี ไม่ควรให้ Collection เกิดการขยายตัวเพราะจะทำให้ความเร็วในการทำงานลดลง

```
Program TestCollection;

Uses Crt,Dos,Objects;

Var
 CP : PCollection;

Procedure InsertItem;
Begin
 CP^.Insert(NewStr('Banana'));
 CP^.Insert(NewStr('Apple'));
 CP^.Insert(NewStr('Peach'));
 CP^.Insert(NewStr('Cherry'));
 CP^.Insert(NewStr('Grapefruit'));
End;

Procedure WriteAllItem;

 Procedure WriteOneItem(P: PString); far;
 Begin
   Writeln(P^);
 End;

Begin
  CP^.ForEach(@WriteOneItem);
End;

Procedure SearchItem;
Var
 Result : PString;
 Key : String;

 Function Quitting: Boolean;
 Var
   ch : Char;

 Begin
 Write('Search agin? ');
   Ch := Upcase(Readkey);
   If Ch<>'Y' then
     Quitting := True
   Else
     Quitting := False;
 End;

 Function Matcheskey(P : PString): Boolean; far;
 Begin
   If Pos(Key, P^)=1 Then
     Matcheskey := True
   Else
     Matcheskey := False;
 End;

Begin
 Writeln; Writeln('Search String');
 Repeat
  Write('Search for?');
  Readln(Key);
  Result := CP^.FirstThat(@MatchesKey);
  if Result <> Nil  Then
    Writeln(Result^)
  Else
    Writeln('***Error String Not Found');
 Until Quitting;
End;

Begin
 CP := New(PCollection, Init(250, 50));
 InsertItem;
 WriteAllItem;
 SearchItem;
 CP^.DeleteAll;
 Dispose(CP, Done);
 Writeln; Writeln('End Program');
End.
```

รูปที่ 3 แสดงโปรแกรม TestCol1.pas

การจัดการข้อมูลใน Collections

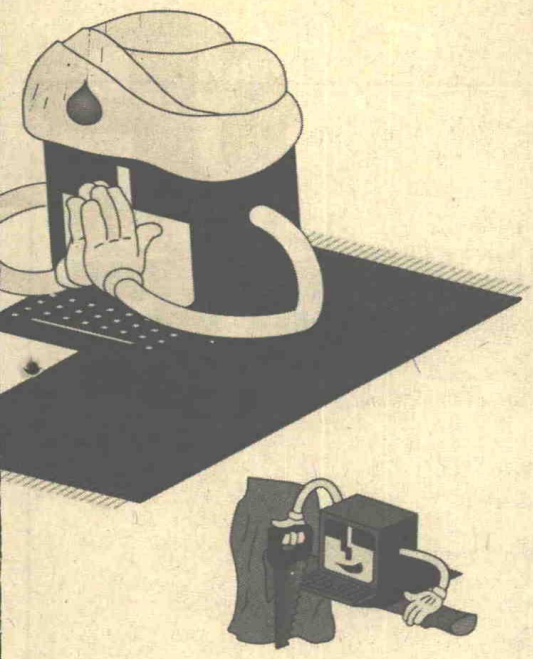
ดังที่ได้กล่าวในตอนต้นแล้วว่า มีวิธีในการใช้งานรายการ Collection Items ถึง 23 วิธี แต่ผมจะแสดงเฉพาะวิธีที่สำคัญ ๆ และเป็นที่นิยมใช้กันเท่านั้น โดยโปรแกรม TestCol1.Pas ในรูปที่ 3 จะแสดงถึงการใช้ TCollections ForEach และ FirstThat function ในการ Browse และ Search Collections ซึ่งแต่ละฟังก์ชันจะเป็นการใช้เทคนิคที่เรียกว่า รูทีนเรียกกลับ (call-back routine) ซึ่งอาจจะเป็นโพรซีเดอร์หรือฟังก์ชันซึ่งจะถูกเรียกโดยอีกรูทีนหนึ่ง ซึ่งคุณจะต้องผ่านค่าแอดเดรสของรูทีนที่คุณจะทำการเรียกไปด้วย

วิธีการ ForEach Method

จากโปรแกรม Testcol1.Pas ในรูปที่ 3 นั้น ให้พิจารณาดู procedure WriteAllItem ซึ่งเป็นคอลแบคโพรซีเดอร์ ทำหน้าที่พิมพ์แสดงข้อมูลทีละไอเท็ม ภายในโพรซีเดอร์ writeAllItem จะมีสเตทเมนต์ดังข้างล่างนี้

```
ColP^.ForEach(@WriteOneItem)
```

Statement ForEach ของ ColP



นั้นจะทำการเรียก WriteOneIem ในการแสดงข้อมูลทีละรายการ ซึ่งจะต้องทำการส่งแอดเดรสของโพรซีเดอร์โดยการใช้ @operator และ WriteOne-Item จะรับค่าพอยน์เตอร์ของแต่ละรายการที่ส่งมา แล้วจะทำการพิมพ์ค่ารายการที่ส่งมาออกทางจอภาพ (อย่าลืมว่าค่ารายการที่ใช้ได้รับเป็นพอยน์เตอร์) ผมคิดว่าถ้าหากดูซอร์สโปรแกรมประกอบจะช่วยให้เข้าใจการทำงานของวิธี ForEach ดียิ่งขึ้น

วิธีการ FirstThat

จะใช้ในการค้นหารายการที่ต้องการในคอลเลคชัน ซึ่งจะใช้วิธีการของโพรซีเดอร์แบบเรียกกลับเหมือนกับวิธี ForEach ฉะนั้นจึงต้องส่งค่าแอดเดรสของสับโพรซีเดอร์โดยใช้ @operator เช่นกัน โปรแกรม TestCol1.pas ในรูปที่ 3 จะเป็นตัวอย่างการใช้งานวิธี FirstThat

จากตัวอย่างจะทำการส่งกลับค่าพอยน์เตอร์ของรายการแรกที่พบตามเงื่อนไขที่กำหนด โดยในที่นี้คอลแบคฟังก์ชันคือ Matcheskey ซึ่งจะทำการเปรียบเทียบว่าถูกต้องตามเงื่อนไขหรือไม่ โดยจะรีเทอร์นค่าเป็นถูก (True) หรือผิด (False) โดยการค้นหาจะเริ่มค้นหาตั้งแต่รายการแรกจนกระทั่งพบค่าที่ต้องการ ก็จะส่งกลับค่าถ้าถูกและ Col-Pâ.FirstThat จะรีเทอร์นค่าพอยน์เตอร์ของรายการที่ค้นพบให้แก่ตัวแปร Result แต่ถ้าหากว่า Matcheskey ฟังก์ชันทำการค้นหาแล้วไม่พบ (กระทั่งถึงรายการสุดท้าย) result ก็จะได้รับค่า nil ภายหลังการเรียก FirstThat เสร็จสิ้นแล้ว

วิธีการ LastThat

ลักษณะการใช้งานเหมือนกับวิธีการ FirstThat ทุกประการ ยกเว้นแต่การค้นหาจะเริ่มจากรายการสุดท้ายก่อน

วิธีการ At

จะส่งกลับค่าพอยน์เตอร์ของรายการ ณ ตำแหน่งอินเด็กซ์ที่ส่งผ่านโดยพารามิเตอร์ (การใช้งานจะคล้ายกับอะเรย์) โดยที่ตำแหน่งรายการแรกในคอลเลคชันคือ ตำแหน่งที่ 0 ซึ่งตัวอย่างการใช้งานตามตัวอย่างข้างล่าง

```
Data := PString(ColP^.At(0))â;
```

โดยสมมติให้ตัวแปร Data มีไทพ์เป็นสตริง สังเกตว่าใช้ PString ในการแปลงจากพอยน์เตอร์ไปเป็นสตริง (อย่าลืมว่าใน TCollection จะส่งกลับค่าเป็นพอยน์เตอร์)

วิธีการ Atdelete

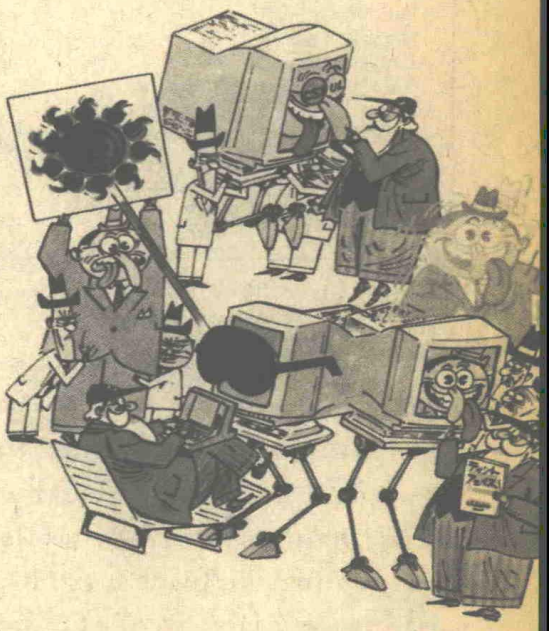
จะใช้ในการลบรายการใน Collection ณ ตำแหน่งที่กำหนด โดยจะต้องส่งพารามิเตอร์เพื่อบอกอินเด็กซ์ที่ต้องการ โดยการใช้งานก็คล้ายกับวิธีการ At

วิธีการ Insert

จะเป็นวิธีที่ใช้ในการแทรกรายการลงในคอลเลคชัน โดยที่รายการที่จะแทรกนั้นจะต้องเป็นพอยน์เตอร์หรือตัวแปร pointer ที่ชี้ไปยังรายการอีกที ตัวอย่างในการใช้งานมีดังนี้

```
Var
      P : PStringCollection; S:Pstring;
begin
      P := New(PstringCollection,Init(10,5));
      S := PString('Test Collection');
      P^.Insert(S);
      P^.Insert(NewStr('Test Insert'));
end;
```

มาถึงตอนนี้แล้วคิดว่าท่านผู้อ่านคงจะพอรู้จักและเข้าใจคอลเลคชันกันบ้างแล้ว เพื่อที่จะทำให้เข้าใจดียิ่งขึ้น จะยกตัวอย่างโปรแกรมง่าย ๆ ไม่ค่อยซับซ้อน ให้ลองไปศึกษา 2 โปรแกรม ซึ่งโปรแกรมแรกชื่อ TestCol1.pas (รูปที่ 3) จะแสดงการ Create, Insert Collection และทำการ Search โดยใช้



```
Program DirCol;
Uses Crt, Objects, Dos;
Type
 PFileCollection = ^TFileCollection;
 TFileCollection = object(TSortedCollection)
  function Compare(Key1, Key2: Pointer): Integer; virtual;
 end;

Var
 FileList: PFileCollection;

function ToUpperStr(P:PString) : string;
var
 i : integer;
 s : string;
begin
  for i := 1 to length(p^) do
    s[i] := upcase(p^[i]);
  s[0] := p^[0];
  ToUpperStr := S
end;

function TFileCollection.Compare(Key1, Key2: Pointer): Integer;
var
 s1, s2 : string;
begin
 s1 := ToUpperStr(PString(Key1));
 s2 := ToUpperStr(PString(Key2));
 if  s1 = s2 then Compare := 0
 else if s1 = '..' then Compare := 1
 else if s2 = '..' then Compare := -1
 else if s1 > s2  then Compare := 1
 else Compare := -1
end;

procedure ReadDirectory(AWildCard: PathStr);
const
 FindAttr = ReadOnly + Archive;
 AllFiles = '*.*';
 PrevDir  = '\..';
var
 S: SearchRec;
 CurPath: PathStr;
 Dir: DirStr;
 Name: NameStr;
 Ext: ExtStr;
 Tmp: PathStr;
begin
 AWildCard := FExpand(AWildCard);
 FSplit(AWildCard, Dir, Name, Ext);
 FileList := New(PFileCollection, Init(5, 5));
 FindFirst(AWildCard, FindAttr, S);
 while (DosError = 0) do
 begin
  if  (S.Attr and Directory = 0) then
    FileList^.Insert(NewStr(S.Name));
  FindNext(S);
 end;
 Tmp := Dir + AllFiles;
 FindFirst(Tmp, Directory, S);
 while (DosError = 0) do
 begin
  if (S.Attr and Directory <> 0) and (S.Name[1] <> '.') then
  begin
    S.Name := '\'+S.Name;
    FileList^.Insert(NewStr(S.Name));
  end;
    FindNext(S);
  end;
  if Length(Dir) > 4 then
  begin
     FindFirst(Tmp, Directory, S);
     FindNext(S);
     if (DosError <> 0) and (S.Name <> PrevDir) then
     begin
       S.Name := PrevDir;
     end;
      S.Name := '\'+S.Name;
     FileList^.Insert(NewStr(S.Name));
  end;
 end;

 var
  s1 : string;
  i,j : integer;
  c : char;
 begin
   clrscr;i := 0;j := 0;
   ReadDirectory('*.pas');
   while i < FileList^.count do
   begin
     s1 := pstring(filelist^.at(i))^;
     writeln(i+1,'  ',s1);
     inc(i); inc(j);
     if j = 20 then
     begin
       writeln('Press any key to continue . . . ');
       c := readkey;
            j := 0;
     end;
   end;
   writeln('Total file : ',i-1);
   readln;
 end.
```

รูปที่ 4 แสดงโปรแกรม DirCol.pas

วิธีการ ForEach ส่วนโปรแกรม DirCol.pas (รูปที่ 4) จะเป็นการนำเอาชื่อไฟล์และไดเรคทอรีมาเก็บใน TSortedCollection ซึ่งผมไม่ได้อธิบายไว้ แต่ว่าคงไม่ยากในการทำความเข้าใจ และในส่วนท้ายของโปรแกรมจะแสดงการค้นหาโดยการใช้วิธีการ At

สรุป

วัตถุประสงค์ของบทความนี้เพื่อเป็นการแนะนำการใช้งานคอลเลคชัน ซึ่งนับเป็นดาต้าสตรัคเจอร์ที่เป็นประโยชน์อย่างมากในการทำงาน ซึ่งความจริงแล้วคอลเลคชันนี้ไม่ใช่เรื่องใหม่ แต่มีอยู่ในเทอร์โบปาสคาลตั้งแต่เวอร์ชัน 6.0 ขึ้นไป ที่อยู่ในเทอร์โบวิชันทูลส์ แต่ว่าเราสามารถนำมาใช้งานโดยไม่ต้องโปรแกรมในรูปแบบ OOP (ที่ได้แนะนำในข้างต้น) ก็ได้